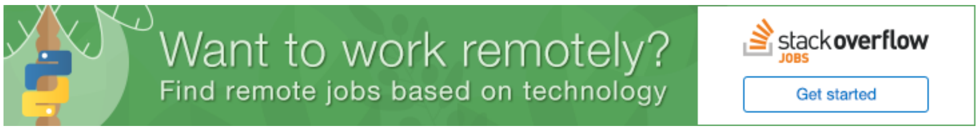
When use ResponseEntity<T> and @RestController for Spring RESTful applications



I am working with Spring Framework 4.0.7, together with MVC and Rest

I can work in peace with:

- `@Controller`
- `ResponseEntity<T>`

For example:

```
@Controller
@RequestMapping("/person")
@Profile("responseentity")
public class PersonRestResponseEntityController {
```

With the method (just to create)

```
@RequestMapping(value="/", method=RequestMethod.POST)
public ResponseEntity<Void> createPerson(@RequestBody Person person, UriComponentsBuilder
ucb){
    logger.info("PersonRestResponseEntityController  - createPerson");
    if(person==null)
        logger.error("person is null!!!");
    else
        logger.info("{}", person.toString());

    personMapRepository.savePerson(person);
    HttpHeaders headers = new HttpHeaders();
    headers.add("1", "uno");
    //http://localhost:8080/spring-utility/person/1
    headers.setLocation(ucb.path("/person/{id}").buildAndExpand(person.getId()).toUri());

    return new ResponseEntity<>(headers, HttpStatus.CREATED);
}
```

to return something

```
@RequestMapping(value="/{id}", method=RequestMethod.GET)
public ResponseEntity<Person> getPerson(@PathVariable Integer id){
    logger.info("PersonRestResponseEntityController  - getPerson - id: {}", id);
    Person person = personMapRepository.findPerson(id);
    return new ResponseEntity<>(person, HttpStatus.FOUND);
}
```

Works fine

I can do the same with:

- `@RestController` (I know it is the same than `@Controller` + `@ResponseBody`)
- `@ResponseStatus`

For example:

```
@RestController
@RequestMapping("/person")
@Profile("restcontroller")
public class PersonRestController {
```

With the method (just to create)

```
@RequestMapping(value="/", method=RequestMethod.POST)
@ResponseStatus(HttpStatus.CREATED)
public void createPerson(@RequestBody Person person, HttpServletRequest request,
HttpServletResponse response){
    logger.info("PersonRestController  - createPerson");
    if(person==null)
        logger.error("person is null!!!");
    else
        logger.info("{}", person.toString());

    personMapRepository.savePerson(person);
    response.setHeader("1", "uno");

    //http://localhost:8080/spring-utility/person/1
    response.setHeader("Location",
request.getRequestURL().append(person.getId()).toString());
}
```

to return something

```
@RequestMapping(value="/{id}", method=RequestMethod.GET)
@ResponseStatus(HttpStatus.FOUND)
```

```
public Person getPerson(@PathVariable Integer id){
    logger.info("PersonRestController  - getPerson - id: {}", id);
    Person person = personMapRepository.findPerson(id);
    return person;
}
```

My questions are:

1. when *for a solid reason* or *specific scenario* one option must be used mandatorily over the other
2. If (1) does not matter, what approach is suggested and why.

spring spring-mvc spring-3 spring-4

asked Oct 24 '14 at 13:56
Manuel Jordan
3,925 8 33 47

3 Answers

`ResponseEntity` is meant to represent the entire HTTP response. You can control anything that goes into it: status code, headers, and body.

`@ResponseBody` is a marker for the HTTP response body and `@ResponseStatus` declares the status code of the HTTP response.

`@ResponseStatus` isn't very flexible. It marks the entire method so you have to be sure that your handler method will always behave the same way. And you still can't set the headers. You'd need the `HttpServletResponse` or a `HttpHeaders` parameter.

Basically, `ResponseEntity` lets you do more.

answered Oct 24 '14 at 15:17
Sotirios Delimanolis
188k 37 399 504

4 Good point about the third observation. Thank You… and I thought the same about `ResponseEntity` , it is more flexible. Just I was with the doubt about `@RestController` . Thank you – Manuel Jordan Oct 24 '14 at 15:25

To complete the answer from Sotorios Delimanolis.

It's true that `ResponseEntity` gives you more flexibility but in most cases you won't need it and you'll end up with these `ResponseEntity` everywhere in your controller thus making it difficult to read and understand.

If you want to handle special cases like errors (Not Found, Conflict, etc.), you can add a `HandlerExceptionResolver` to your Spring configuration. So in your code, you just throw a specific exception (`NotFoundException` for instance) and decide what to do in your Handler (setting the HTTP status to 404), making the Controller code more clear.

edited Apr 19 '17 at 14:28
cherit
5,104 6 35 68

answered Apr 30 '15 at 10:55
Matt
1,151 8 19

3 Your point of view is valid working with (@)ExceptionHandler. The point is: if you want all handled in one method (Try/Catch) HttpEntity fits well, if you want reuse exception handling (@)ExceptionHandler for many (@)RequestMapping fits well. I like HttpEntity because I am able to work with HttpHeaders too. – Manuel Jordan Apr 30 '15 at 12:54

According to official documentation: Creating REST Controllers with the @RestController annotation

> @RestController is a stereotype annotation that combines @ResponseBody and @Controller. **More than that, it gives more meaning to your Controller and also may carry additional semantics in future releases of the framework.**

It seems that it's best to use `@RestController` for clarity, but you can also **combine** it with `ResponseEntity` for flexibility when needed (According to official tutorial and the code here and my question to confirm that).

For example:

```
@RestController
public class MyController {

    @GetMapping(path = "/test")
    @ResponseStatus(HttpStatus.OK)
    public User test() {
        User user = new User();
        user.setName("Name 1");

        return user;
    }

}
```

is the same as:

```
@RestController
public class MyController {

    @GetMapping(path = "/test")
    public ResponseEntity<User> test() {
        User user = new User();
        user.setName("Name 1");

        HttpHeaders responseHeaders = new HttpHeaders();
        // ...
        return new ResponseEntity<>(user, responseHeaders, HttpStatus.OK);
    }

}
```

This way, you can define `ResponseEntity` only when needed.

Update

You can use this:

```
    return ResponseEntity.ok().headers(responseHeaders).body(user);
```

edited Nov 14 '17 at 9:45
Mona Mohamadinia
17 5

answered Nov 6 '16 at 21:29
Danail
624 7 14

What if we have added @ResponseStatus(HttpStatus.OK) on the method, but method returns return new ResponseEntity<>(user, responseHeaders, HttpStatus.NOT_FOUND); I am just thinking that whether @ResponseStatus will modify the response code further. – Hemant Patel Dec 5 '16 at 6:08

2 @Hemant seems that `@ResponseStatus(HttpStatus.OK)` is ignored when you return `ResponseEntity<>(user, responseHeaders, HttpStatus.NOT_FOUND)`. The HTTP response is `404` – Danail Dec 7 '16 at 21:13